МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторної роботи 6
за темою «Використання вказівників на функції та заголовних файлів»
з курсу «Алгоритмізація та програмування. Частина 1»

для студентів спеціальності
122 «Комп’ютерні науки»

Харків 2019

Методичні вказівки до виконання лабораторної роботи 6 за темою «Використання вказівників на функції та заголовних файлів» з курсу «Алгоритмізація та програмування. Частина 1» для студентів спеціальності 122 «Комп’ютерні науки» / уклад. Л. В. Іванов, М. О. Білова. – Харків : НТУ «ХПІ», 2019. – 19 с.

Укладачі: Л. В. Іванов
М. О. Білова

Рецензент О. В. Шматко

Кафедра програмної інженерії та інформаційних технологій управління

ЗМІСТ

Вступ

Курс «Алгоритмізація та програмування» присвячений теоретичним та практичним аспектам розробки алгоритмів і програм мовою С++. Отримані в результаті вивчення даної дисципліни знання та навички можуть бути використані в усіх наступних курсах, під час виконання курсових та дипломних проектів, вивчення дисциплін, що пов'язані з обчислювальною технікою та програмуванням.

У процесі виконання лабораторної роботи за темою «Використання вказівників на функції та заголовних файлів» студенти мають закріпити теоретичний матеріал, отримати навички практичної роботи при реалізації програм, пов'язаних зі вказівниками.

Перша частина стосується використання псевдонімів типів, вказівників на функції, заголовних файлів. Надано визначення та сформульовані приклади використання стражів включення, розглянуто особливості роботи з просторами імен. У практичній частині подано код двох програм: використання заголовних файлів і пошуку кореня на відрізку за методом дихотомії.

Друга частина складається з завдань для лабораторної роботи, що використовуються як поточний контроль засвоєння матеріалу. Лабораторні заняття розраховані на роботу в комп'ютерному класі під безпосереднім керівництвом викладача та самостійну роботу студентів, яка передбачає написання тексту програми, закріплення практичних навичок роботи на комп'ютері, набутих при виконанні відповідної лабораторної роботи. Варіанти індивідуальних завдань видаються викладачем на занятті.

У методичних вказівках подано два завдання. У цілому методичні вказівки будуть корисні студентам різних спеціальностей і форм навчання, які вивчають мову С ++.

ТЕОРЕТИЧНА ЧАСТИНА

1. Псевдоніми типів

C++ дозволяє створити псевдонім для існуючого імені типу, використовуючи ключове слово `typedef`. У результаті створюється синонім для типу. Важливо відрізняти створення синоніма від створення нового типу (визначення структур, перелічень і класів). Визначення синоніму починається з ключового слова `typedef`, за яким розташовують існуючий тип, а потім – нове ім'я (ідентифікатор). Наприклад,

```
typedef unsigned long int Cardinal;
typedef int IntArray[15];
```

створює нове ім'я `Cardinal`, яке можна використовувати у будь-якому місці замість `unsigned long int`. Ідентифікатор `IntArray` може бути використаний для визначення масиву з 15 цілих значень:

```
Cardinal c;
int f(Cardinal k);
IntArray a; // int a[15];
```

Визначення `typedef` інтерпретується таким же чином, як визначення змінної, але ідентифікатор стає синонімом типу.

Визначення `typedef` дозволяє побудувати більш короткі або більш значущі імена для типів, які вже визначені в мові або для користувацьких типів. Ці імена дозволяють інкапсулювати деталі реалізації, які можуть змінитися.

2. Вказівники на функції

Вказівник на функцію – це адреса, де зберігається скомпільований код цієї функції, тобто адреса, за якою передається управління, коли ця функція викликається. Так само, як ім'я масиву є константним вказівником

на перший елемент масиву, ім’я функції можна розглядати як константний вказівник на функцію. Можна оголосити змінну – вказівник, який вказує на функцію, і викликати функцію за допомогою цього вказівника.

Вказівник на функцію повинен вказувати на функцію з відповідним типом результату і сигнатурою. У визначенні

```
int (*funcPtr)(double);
```

`funcPtr` оголошується вказівник, який вказує на функцію, що приймає параметр з рухомою комою і повертає ціле значення. Дужки навколо `*funcPtr` необхідні. Без першої пари дужок це був би прототип функції, яка приймає `double` і повертає вказівник на `int`. Оголошення вказівника на функцію завжди буде включати в себе тип значення і круглі дужки, які вказують типи параметрів.

Вказівнику на функцію можна присвоїти адресу певної функції шляхом присвоєння імені функції без дужок. Тоді вказівник можна використовувати так само, як ім’я функції. Вказівник на функцію повинен бути узгоджений з функцією за типом результату і сигнатурою. Наприклад:

```
int round(double x)
{
    return x + 0.5;
}

void main()
{
    int (* funcPtr)(double);
    double y;
    cin >> y;
    funcPtr = round;
    cout << funcPtr(y);
}
```

Вказівник на функцію не потрібно розіменовувати, хоча це можна зробити. Тому, якщо `pFunc` є покажчиком на функцію, в який записано адресу реальної функції, можна викликати цю функцію безпосередньо

```
pFunc(x);
```

або

```
(*pFunc)(x);
```

Обидві форми ідентичні.

Для створення більш відповідних імен типів вказівників на функції часто використовують визначення `typedef`:

```
typedef int (*FuncType)(int);
FuncType pf;
```

Можна створити масив указівників на функції. Вказівники на функції часто використовуються як типи аргументів функцій.

3. Використання заголовних файлів

Кожна нетривіальна програма може бути розділена на відносно універсальні частини, які можуть бути використані у кількох проектах, та частини, специфічні для конкретного проекту. Універсальні та специфічні частини доцільно зберігати в окремих файлах.

Найпростіший шлях для реалізації цієї ідеї полягає у застосуванні директиви препроцесору `#include`, яка дозволяє перед компіляцією включити вихідний текст з одного файлу всередину іншого.

Препроцесор не здійснює фізичного копіювання змісту файлу. Замість цього препроцесор створює новий вихідний текст у оперативній пам’яті. Цей

текст має назву одиниці трансляції. Можна також видалити певні частини вихідного тексту за допомогою директив `#define`, `#ifdef` та `#ifndef`. Директива `#define` дозволяє визначити нову змінну препроцесору в будь-якому місці. В іншому місці можна перевірити цей факт за допомогою директив `#ifdef` або `#ifndef`. Наприклад:

```
#define New_Name
...
#ifdef New_Name
// Включаємо цей код в одиницю трансляції
#else
// Не включаємо цього коду в одиницю трансляції
#endif
```

Одиниця трансляції – це результат обробки вихідного тексту препроцесором. Один проект може містити кілька одиниць трансляції. Імена, визначені в одних одиницях, повинні бути об’явлені в інших одиницях. Неправильне оголошення імені може привести до помилок. Правильним рішенням є створення окремого **заголовного файлу** з необхідними описами. Включення заголовних файлів гарантує точне відтворення об’яв у всіх одиницях трансляції.

У заголовний файл можна включати такі елементи:

– іменовані простори імен;

– визначення типів;

– об’яви функцій;

– визначення функцій з модифікатором `inline`;

– об’яви даних (з ключовим словом `extern`);

– визначення констант;

– директиви препроцесору;

– коментарі.

Заголовні файли не повинні містити:

– визначень звичайних функцій;

– визначень даних;

– безіменних просторів імен.

Традиційно заголовні файли мають розширення `.h`, тоді як файли з реалізацією функцій та визначенням даних мають розширення `.cpp`.

Існують чисельні стандартні заголовні файли з описами класів та функцій. Для включення таких файлів замість лапок слід вживати кутові дужки <> для того, щоб препроцесор шукав такі файли в стандартних теках. В іншому випадку препроцесор починає пошук з поточної теки.

4. Стражі включення (Include Guards)

Дуже часто у вихідний текст необхідно включати більше одного заголовного файлу. Крім того, може існувати необхідність включення тексту одного заголовного файлу в інший заголовний файл. Наприклад, необхідність включення файлу `f1.h` у файл `f2.h`, при тому, що файл `f3.h` потребує включення файлів `f1.h` та `f2.h`, а всі ці файли необхідні для компіляції основної програми:

```
//f1.h
...

//f2.h
#include "f1.h"
...

//f3.h
#include "f1.h"
#include "f2.h"
...

//main.cpp
#include "f1.h"
#include "f2.h"
#include "f3.h"
...
```

Препроцесор включає вміст файлу `f1.h` в одиницю трансляції, коли він обробляє файл `main.cpp`. Потім він включає вміст файлу `f2.h`, який також містить включення `f1.h`. Після включення `f3.h` одиниця трансляції містить чотири копії тексту `f1.h` та дві копії `f2.h`. Таке включення не має сенсу, а також може викликати помилки під час компіляції.

Найчастіше цю проблему вирішують за допомогою так званих **стражів включення** – спеціальної конструкції з директив препроцесору. Наприклад, текст заголовного файлу `f1.h` можна організувати в такий спосіб:

```
#ifndef F1_H
#define F1_H
... // the whole file goes here
#endif
```

Коли у програмі вперше згадується включення заголовного файлу, препроцесор здійснює читання першого рядку (`#ifndef`) та включає весь текст у одиницю трансляції, оскільки змінна `1F1_H` ще не була визначена. Окрім того, препроцесор визначає цю змінну.

Вдруге та будь-коли потім препроцесор не включає тексту заголовного файлу, оскільки змінна `F1_H` вже визначена.

Ім’я, яке визначається у директиві `#define`, може бути довільним. Важливо тільки, щоб воно було визначене тільки в одному заголовному файлі. В іменах змінних препроцесору заборонено застосовувати крапку.

5. Створення заголовних файлів у середовищі Microsoft Visual Studio .NET

Необхідні кроки полягають у такому:

– вибрати `Add New Item` у підменю `Project`;

– вибрати `Header File (.h)` у вікні `Templates`;

– ввести нове ім’я файлу без розширення у полі `I Name`;

– натиснути кнопку `Add`;

Можна додати новий файл реалізації аналогічним чином.

6. Простори імен

Простори імен визначають логічну структуру програми.

Простір імен являє собою іменовану область, яка може містити оголошення і визначення констант, змінних, функцій і типів, а також інших просторів імен. Простори імен дозволяють уникнути конфліктів імен. Простори імен дають механізм для логічного групування оголошень і визначень.

```
namespace MySpace
{
    int  k = 10;
    void f(int n)
    {
        k = n;
    }
}
```

Елементи просторів імен можуть бути визначені окремо від своїх оголошень. Наприклад,

```
namespace MySpace
{
    int  k = 10;
    void f(int n);
}
void MySpace::f(int n)
{
    k = n;
}
```

Простори імен можуть бути вкладені в інші простори імен:

```
namespace FirstSpace
```

```
{
    namespace SecondSpace
    {
        ...
    }
}
```

Можна використовувати альтернативне ім’я для ідентифікації простору імен:

```
namespace YourSpace = MySpace;
```

Простори імен можна переривати й відкривати знову для подальшого розширення:

```
namespace FirstSpace
{
    // перша частина
}

... // інші описи
namespace SecondSpace
{
    // інший простір імен
}
namespace FirstSpace
{
    // друга частина
}
```

Оголошення одного простору імен можуть знаходитися в різних файлах.

Є три способи доступу до елементів простору імен:

– за допомогою явної кваліфікації доступу;

– за допомогою using-оголошення;

– за допомогою using-директиви.

Перший варіант дозволяє дістатися кожного імені через ідентифікатор простору і необхідне ім’я всередині простору, розташоване після операції дозволу видимості (::). Наприклад:

```
int x = MySpace::k;
```

Другий варіант дозволяє отримати доступ до членів простору імен в індивідуальному порядку з використанням синтаксису using-оголошення. Оголошений ідентифікатор додається до локального простору імен:

```
using MySpace::k;
using MySpace::f;
int y = k + f(k);
```

Третій спосіб використовують, якщо є потреба у використанні кількох (або всіх) членів простору імен. За допомогою using-директиви визначають, що всі ідентифікатори у просторі імен знаходяться у глобальній області видимості, починаючи з точки, де визначена using-директива. Наприклад:

```
using namespace MySpace;
```

Слід уникати використання цієї директиви через можливі конфлікти імен.

Директива `using` може бути використана, якщо необхідно кілька просторів імен:

```
namespace NewSpace
{
  using namespace FirstSpace;
  using namespace SecondSpace;
}
```

Можна вибрати кілька імен з одного або декількох просторів імен у новому просторі імен за допомогою using-оголошення:

```
namespace NewSpace
{
  using OtherSpace::name1;
  using OtherSpace::name2;
}
```

Такий простір імен може бути використаний у різних проектах.

Простори імен у C++ не приховують дані. Після того як простір імен було підключено за допомогою директиви using, усі імена, оголошені у просторі імен, можуть бути використані як глобальні імена без жодних обмежень.

Більшість компонентів стандартної бібліотеки C++ згрупована у просторі імен `std`. Простір імен `std` містить додаткові простори імен, такі як, наприклад, `std::rel_ops`.

ПРИКЛАДИ ПРОГРАМ

1. Метод ділення відрізку навпіл (дихотомії). Наведена нижче програма знаходить корінь рівняння за допомогою методу дихотомії. Єдине обмеження використання методу дихотомії полягає у тому, що рівняння повинно мати рівно один корінь на заданому інтервалі.

```
#include <iostream>
#include <cmath>
using std::cout;
using std::endl;
using std::fabs;

typedef double (*FuncType)(double);

// Четвертий аргумент має усталене значення:
double root(FuncType f, double a, double b, double eps =
0.001)
{
```

```
    double x;
    do
    {
        x = (a + b) / 2;
        if (f(a) * f(x) > 0)
        {
            a = x;
        }
        else
        {
            b = x;
        }
    }
    while (b - a > eps);
    return x;
}

double g(double x)
{
    return x * x - 2;
}

void main()
{
    cout << root(g, 0, 6) << endl;
    cout << root(g, 0, 6, 0.00001) << endl;
    cout << root(sin, 1, 4) << endl;
    cout << root(sin, 1, 4, 0.00001) << endl;
}
```

2. Використання заголовних файлів. Нехай необхідно створити одиницю трансляції, побудовану з заголовного файлу `SomeFile.h` і файлу реалізації `SomeFile.cpp`.

Спочатку додають стражі включення:

```
#ifndef SomeFile_h
#define SomeFile_h
#endif
```

У заголовному файлі перед `#endif` розташовують прототип функції:

```
#ifndef SomeFile_h
#define SomeFile_h
int sum(int a, int b);
```

```
#endif
```

Файл реалізації містить визначення функції.

```
#include"SomeFile.h"
int sum(int a, int b)
{
    return a + b;
}
```

Тепер заголовний файл `SomeFile.h` слід включити у головну програму:

```
#include <iostream>
#include "SomeFile.h"

using namespace std;

void main()
{
    int x, y;
    cout << "Enter two integer values:" << endl;
    cin >> x >> y;
    int z = sum(x, y);
    cout << "Sum is " << z << endl;
}
```

ВПРАВА ДЛЯ КОНТРОЛЮ

Реалізувати приклади і вправи лабораторної роботи 3 за темою «Використання функцій» з курсу «Алгоритмізація та програмування. Частина 1» з розташуванням в окремій одиниці трансляції всіх функцій, крім `main()`.

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ

1. Метод ділення відрізку навпіл

Реалізувати програму знаходження кореня рівняння методом ділення відрізку навпіл (дихотомії) у різних одиницях трансляції. Описати та використати вказівник на функцію, тип якого, а також прототип функції знаходження кореня слід розташувати у заголовному файлі.

2. Індивідуальне завдання

Написати програму, яка реалізує перебір значень з метою пошуку деякого значення відповідно до індивідуального завдання, наведеного в табл. 7. Необхідне значення може бути знайдено шляхом перевірки проміжних значень функції (або першої / другої похідної). Слід використати вказівник на функцію, для якого визначити `typedef`.

Сирцевий код повинен бути розділений на дві одиниці трансляції. Перша одиниця трансляції буде представлена заголовним файлом і файлом реалізації. Визначення `typedef`, а також прототип функції пошуку потрібного значення, повинні бути розташовані в заголовному файлі. Визначення цієї функції слід здійснити у файлі реалізації. Функція для перевірки працездатності програми, а також функція `main()`, повинні бути розташовані в іншій одиниці трансляції.

Таблиця 7 – Варіанти завдань для лабораторної роботи

Номер варіанта (номер студента у списку)	Правило пошуку
1	Максимальне значення другої похідної
2	Мінімальне значення першої похідної
3	Найменший корінь
4	Найбільший корінь
5	Сума мінімального і максимального значень
6	Добуток мінімального і максимального значень
7	Кількість коренів
8	Найменший корінь другої похідної
9	Мінімальне значення другої похідної
10	Максимальне значення першої похідної
11	Найменший корінь першої похідної
12	Найбільший корінь першої похідної
13	Найбільший корінь другої похідної
14	Сума мінімального і максимального значень

Функція для тестування може бути довільною, зокрема можна використати функцію із завдання першої лабораторної роботи.

Примітка: для обчислення першої похідної $y(x)$ можна використати таку формулу:

$$y'(x) = \frac{y(x + \Delta x) - y(x)}{\Delta x},$$

де Δx – деяке невелике значення, наприклад 0,0000001.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Як створити синонім для вже існуючого типу?
2. Що таке вказівник на функцію?
3. Для чого використовують вказівники на функції?
4. Як визначити вказівник на функцію?
5. Що таке одиниця трансляції?
6. Як здійснюється використання директиви `#define`?
7. Які правила розподілу сирцевого коду між заголовним файлом і файлом реалізації?
8. У чому різниця між включенням стандартних заголовних файлів і заголовних файлів користувача?
9. Що таке стражі включення?
10. Що таке простір імен?
11. Як об’єднати кілька просторів імен в один?
12. Як визначити псевдонім для існуючого простору імен?

Рекомендована література

1. Bjarne Stroustrup. The C++ Programming Language. Third Edition / Bjarne Stroustrup. – Addison-Wesley, 1997.

2. Stanley B. Lippman C++ Primer. Third Edition / Stanley B. Lippman, Josee Lajoie. – Addison-Wesley, 1988.

3. Deitel H. M. C++. How to Program. Third Edition / H. M. Deitel, P. J. Deitel. – Prentice Hall, 2001.

4. Голуб Б. М. C#. Концепція та синтаксис / Б. М. Голуб. – Львів: Видавничий центр ЛНУ ім. Івана Франка, 2006. – 136 с.

5. Грицюк Ю. І. Програмування мовою С++: навч. посібник / Ю. І. Грицюк, Т. Є. Рак. – Львів : Вид-во Львівського ДУ БЖД, 2011. – 292 с.

Internet-джерела

1. The C++ Programming Language (Bjarne Stroustrup's homepage) // http://www2.research.att.com/~bs/C++.html

2. ISO/IEC 14882:2003 Programming languages - C++ (International Stamdard) // http://cs.nyu.edu/courses/summer12/CSCI-GA.2110-001/downloads/C++%20Standard%202003.pdf

3. The C++ Resources Network // http://www.cplusplus.com/

4. The C++ Tutorial // http://www.learncpp.com/

5. C++ – Вікіпідручник // http://uk.wikibooks.org/wiki/C++

6. C++ – Вікіпіедія // http://uk.wikipedia.org/wiki/C++

7. Корх О. Основи мови програмування С++ // http://korkholeh.googlepages.com/cppfund.pdf

Навчальне видання

Методичні вказівки

до виконання лабораторної роботи 6

за темою «Використання вказівників на функції та заголовних файлів»

з курсу «Алгоритмізація та програмування. Частина 1»

для студентів спеціальності

122 «Комп’ютерні науки»

Укладачі:

ІВАНОВ Лев Вадимович

БІЛОВА Марія Олексіївна

Відповідальний за випуск М. Д. Годлевський

Роботу до видання рекомендував О. В.Горілий

План 2018 р., поз. 315

Підписано до друку 28.05.2019. Гарнітура Times New Roman.

Ум. друк, арк. 0,8.

Видавничий центр НТУ «ХПІ»,

вул. Кирпичова, 2, м.Харків-2, 61002

Свідоцтво про державну реєстрацію ДК № 3478 від 21.08.2017 р.

Самостійне електронне видання